Byzantine-tolerant Uniform Node Sampling Service in Large-scale Networks

Emmanuelle Anceaume, Yann Busnel, Bruno Sericola

Byzantine-tolerant Uniform Node Sampling Service in Large-scale Networks

Emmanuelle Anceaume [a] and Yann Busnel [b] and Bruno Sericola [c]

[a] CNRS, Univ Rennes, Inria, IRISA, Rennes, France;
[b] IMT Atlantique, IRISA, Cesson-Sévigné, France;
[c] Inria, Univ Rennes, CNRS, IRISA, Rennes, France

ARTICLE HISTORY
Compiled March 15, 2021

ABSTRACT
We consider the problem of achieving uniform node sampling in large scale systems in presence of Byzantine nodes. The uniform node sampling service offers to applications using it a single simple primitive that returns, upon invocation, the identifier of a random node that belongs to the system. We first propose an omniscient strategy that processes on the fly an unbounded and arbitrarily biased input stream made of node identifiers exchanged within the system, and outputs a stream that preserves the uniformity property. Informally, uniformity states that any node in the system should have the same probability to appear in the sample of any correct node of the system. We show through a Markov chain analysis that this property holds despite any arbitrary bias introduced by the adversary. We then propose a strategy based on a sketch data structure that is capable of approximating the omniscient strategy without requiring any prior knowledge on the composition of the input stream. We show through both theoretical analysis and extensive simulations that this "knowledge-free" strategy accurately approximates the omniscient one. We evaluate the resilience of the knowledge-free strategy by studying two representative attacks (flooding and targeted attacks). We quantify the minimum number of identifiers that Byzantine nodes must insert in the input stream to prevent uniformity. Finally, we propose a new construction that processes each input stream with sketches put in series that allows to both increase the accuracy of a single sketch and decrease the time to converge to a uniform output stream. To our knowledge, such a work has never been proposed before.

KEYWORDS
Data stream; Byzantine nodes; Uniform sampling; Markov chains;Randomized approximation algorithm

The uniform node sampling service offers to applications using it a single simple primitive that returns the identifier of a random node that belongs to the system. Providing at any time randomly chosen nodes in the system has deserved a lot of attention to construct large scale distributed applications. A typical example is load

This work has been partially founded by the French ANR project SocioPlug (ANR-13-INFR-0003) and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

A shorter preliminary version of this paper entitled "Uniform Node Sampling Service Robust against Collusions of Malicious Nodes" appeared in the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013.

CONTACT Emmanuelle Anceaume. Email: emmanuelle.anceaume@irisa.fr

balancing in cluster-based applications: choosing a host at random among those that are available is often a choice that provides performance close to that offered by more complex selection criteria, without imposing any burden [1]. Another case where having access to random node identifiers is important is epidemic-based applications: by periodically selecting few random nodes as neighbours, large-scale environments preserve their connectivity despite node dynamicity [2–5].

Unfortunately, the unavoidable presence of malicious nodes in large scale and open systems seriously impedes the construction of uniform node sampling [6–8]. The objective of malicious nodes mainly consists in continuously and largely biasing the input data stream out of which samples are obtained, to prevent (correct) nodes from being selected as samples. Consequences of these collective attacks are, among others, *(i)* the overwhelming load of some specific nodes when it is used to provide random locations for data caching or storage, or *(ii)* the eventual partitioning of the system when the node sampling service contributes to the construction of nodes local views in epidemic-based protocols. Solutions that basically consist in storing the identifier of all the nodes of the system so that each of these node identifiers can be randomly selected when needed are impracticable and even infeasible due to the size of the system and the churn, that is the frequent arrival and departure of nodes to and from the system [9]. Rather, providing a solution that requires as little space as possible (*e.g.*, sub-linear in the population size of the system) is definitely desirable. Bortnikov *et al.* [10] have recently proposed a uniform node sampling algorithm that tolerates malicious nodes by exploiting the properties offered by min-wise permutations. Specifically, their sampling component outputs the node identifier whose image value under the randomly chosen permutation is the smallest value ever encountered [10]. Thus eventually, by the property of min-wise permutation, the sampler converges towards a random sample. However by the very same properties of min-wise permutation functions, once the convergence has been reached, it is stuck to this convergence value independently from any subsequent input values. Thus the sample does not evolve according to the current composition of the system, which makes it static. Actually it has been shown in [9] that imposing strict restrictions on the number of messages sent by malicious nodes during a given period of time and providing each correct node with a very large memory (proportional to the size of the system) is a necessary and sufficient condition to output an unbiased and non static stream. Thus, lack of adaptivity or full-space algorithms seem to be the only defenses against adversarial behaviors when considering deterministic algorithms. In this paper, we solve this problem by adopting a probabilistic approach.

Our contributions

The algorithms we propose are designed in the data stream model. In this model, presented in Section 2, nodes of the system receive a possibly infinite sequence of data items, and must process them sequentially and in a single pass. The size of the working memory of each node is sub-linear in n and m, where m represents the number of data items read so far by a node, and n is the size of the set from which data items are drawn. Within this model, we assume the presence of an adversary capable of actively tampering with the data stream of any (correct) node to bias the samples output by this node.

Based on this model, we propose in Section 3 a formal definition of the node sampling service tolerant to malicious nodes. This property says that for any node identifier

present in the input stream, the probability that this node identifier is selected as a sample is equal to $1/n$, where n is the size of the population.

We present a first solution to this problem by assuming that each node exactly knows the probability with which the received node identifiers will occur in its input stream. This solution, called in the following the "*omniscient algorithm*" and presented in Section 4, has only access to a memory of small and constant size. We study the behavior of this algorithm through a Markov chain analysis, and show that it is capable of tolerating any bias, introduced by the adversary, in the input stream of each correct node of the system. We study both the asymptotic and the transient behaviors of the Markov chain.

We then propose in Section 5 a randomized approximation algorithm, called in the following the "*knowledge-free algorithm*", that outputs an almost uniform stream whose deviation from an exact uniform stream is bounded with any tunable probability. This is achieved without any prior knowledge on the composition of the input stream. This algorithm is a one-pass algorithm (*i.e.*, each piece of data of the input stream is scanned sequentially on the fly), and is space-efficient (*i.e.*, only a compact synopsis or sketch that contains the most important information about data items is locally maintained). We show that using $\mathcal{O}(\log(n)\log(1/\delta)/\varepsilon)$ bits of space allows us to additively (ε, δ)-approximate a uniform and fresh output stream from an arbitrarily biased input one.

We then evaluate in Section 6 the minimum effort that needs to be exerted by an adversary to bias the output stream when two representative attacks are launched. The first one, known as a *targeted attack* consists for the adversary to bias the frequency of a single node identifier, while the *flooding attack* aims at biasing all the node identifier frequencies. Both evaluations are conducted by modeling them as an urn problem. One of the main results of this analysis is the fact that the effort that needs to be exerted by the adversary to subvert the sampling service can be made arbitrarily large by any correct node by just increasing the memory space of the sampler.

Extensive simulations achieved with both real data and synthetic traces confirm the robustness of our sampler service. Main results are presented in Section 7.

Finally, we propose in Section 8 a new construction that processes the input stream with sketches put in series. We show that this construction both increases the accuracy of a single sketch and decreases the time needed to converge to a uniform stream, and this is achieved without requiring any additional space nor additional operations per item.

1. Related Work

The authors in [6–8] propose techniques to detect and exclude malicious nodes by observing that malicious nodes try to get an in-degree much higher than correct nodes in order to isolate them. In particular, Jesi *et al.* [6] propose a solution that supposes that the ultimate goal of the malicious nodes is to mutate the random graph into a hub-based graph, hub for which malicious nodes gain the lead. Once this goal is reached, malicious peers can very quickly and easily subvert the whole overlay by performing denial-of-service attacks. Conducting a hub attack mainly consists for malicious peers in increasing their in-degree. Jesi *et al.* [6] propose to detect highly popular peers by extending classic membership services with a module that identifies and blacklists peers that have an in-degree much higher than the other peers of the overlay. This approach, also adopted in several structured based systems [8] through auditing mechanisms, or

in sensor networks [7], is effective only if the number of malicious nodes is very small with respect to the size n of the system (*i.e.*, typically of $O(\log n)$). When the system is populated by a large number of malicious peers (*i.e.*, a linear proportion of the nodes of the system as assumed in our paper), which is definitively a realistic assumption in peer-to-peer systems [11,12], additional mechanisms have been proposed.

In structured peer-to-peer systems, analytical studies have shown that regularly pushing nodes to random positions in the overlay is a necessary and sufficient condition to defend the system against adversarial behaviours [13–15]. By taking advantage of the properties of structured graphs, the authors of both papers [13,14] have shown that, with high probability, any node is equally likely to appear in the local view of each other correct node in a number of rounds polynomial in the size of the system.

Bortnikov *et al.* [10] have proposed a membership algorithm that relies on a sampling service that exploits the properties offered by min-wise permutations. Namely, their sampling service, similarly to the one we propose, is fed with the stream of node identifiers periodically gossiped by nodes. Differently from our solution, their sampler outputs the node identifier whose image value under the randomly chosen permutation is the smallest value ever encountered. Thus eventually, by the property of min-wise permutation, the sampler service converges towards a random sample. By limiting the number of node identifiers malicious nodes can periodically issue (no more than 20% of the total number of requests can be sent by malicious nodes), their solution requires a single node identifier to be stored in the local memory. However, once convergence has been reached, it is stuck to this convergence value independently from the input values. Thus the sample does not evolve according to the current composition of the system. In contrast, and as will be demonstrated in the following, the uniform node sampling algorithm we propose self adapts to the current composition of the system, and as such could be integrated in the membership algorithm proposed by Bortnikov *et al.* [10] to cope with dynamic environment.

Streaming algorithms have shown their highly desirable properties in data intensive monitoring applications [16,17]. These algorithms process the input stream in a single pass and sequentially. All these algorithms rely on pseudo-random functions that map elements of the stream to uniformly distributed image values. The interested reader is invited to read the nice survey by Muthukrishnan [18]. Most of the research done so far with this approach has mainly focused on computing functions or statistic measures with a given small error using a little amount of space with respect to the size of the input stream and the set from which items belong to. These include, as examples, the computation of the number of different data items in a given stream [19–21], the frequency moments [22], the most frequent data items [22,23], the entropy of the stream [24,25], or the proposition of a metric that allows to estimate a broad class of distance measures between any two massive streams [26], or the correlation among different streams [27].

In this work, we go a step further by continuously computing, in an adversarial context, a uniform sample of the nodes of the system so that for any node identifier present in the input stream, the probability that this node identifier is selected as a sample is equal to $1/n$, where n is the size of the population.

2. System Model and Assumptions

2.1. Model of the Network

We consider a large scale and dynamic open system $\mathcal{N}$. Each node of $\mathcal{N}$ is assigned a random identifier from an b-bit identifier space. Identifiers (denoted by ids in the following) are derived by using some standard cryptographic one-way hash function (*e.g.*, SHA256 hash function). The value of b is large enough to make the probability of identifiers collision negligible. Nodes of the system can freely join and leave the system as often as they wish. At any time, each node $i \in \mathcal{N}$ knows only a small set of the nodes of the system with which it can communicate. This set, whose size is classically logarithmic in the total population size, is called the view or the neighborhood of i. To keep the system connected despite churn, each node i periodically renews its view membership. This is achieved through gossip-based algorithms during which node i exchanges some of its neighbors with those of its neighbors.

2.2. Adversary

We assume the presence of malicious (*i.e.*, Byzantine) nodes that collectively try to subvert the system by manipulating the prescribed protocol. A node present in the system that adheres to the protocol is called *correct*. We model adversarial behaviors through an adversary that fully controls and manipulates these malicious nodes. Specifically, we suppose that the adversary owns t node identifiers, with t any positive integer. Each of these t node identifiers does not need to correspond to a single real node. Indeed, the adversary will augment its power by owning numerous node identifiers, such that only a limited number of real malicious nodes are linked to these identifiers. However, affecting multiple identifiers per node is costly as it requires to interact with a central trusted authority to receive a certificate assessing the validity and integrity of each identifier. The adversary may actively tamper with the input stream of any correct node i by sending to i as many messages as possible such that for each of those messages, the adversary uses one of its t node ids as the source address of the message. Formally, each time a correct node i receives a message in its input stream, with probability α, this message has been sent by the adversary, and with probability $1 - \alpha$, it has been sent by a correct node. We will show that t only depends on the sampling protocol parameters, while α may be close to 1. Classically, we assume that the adversary can neither drop a message exchanged between two correct nodes nor tamper with its content without being detected [10]. This is achieved by assuming the existence of a signature scheme (and the corresponding public-key infrastructure) ensuring the authenticity and integrity of messages. This refers to the *authenticated Byzantine failure model* [28]. Note that correct nodes cannot *a priori* distinguish correct nodes from malicious ones. We finally suppose that any algorithm run by any correct node is public knowledge to avoid some kind of security by obscurity.

2.3. Sampling Assumptions

We first assume that there exists a time T_0 such that after that time, the churn of the system ceases (churn is classically defined as the rate of turnover of nodes in large scale systems [29]). This assumption, also adopted by Bortnikov *et al.* [10] is necessary to make the notion of uniform sample meaningful. Thus from T_0 onwards, the population of the system $\mathcal{N}$ is composed of n nodes such that t of them are malicious, with $t < n$.

Note that even if in practice there are not t malicious nodes, but t node identifiers owned by the set of malicious nodes, without loss of generality we assume that those t node identifiers correspond to t real nodes. The value of both t and n is not known by correct nodes. We also suppose that at any discrete time $k \geq T_0$ all the nodes in $\mathcal{N}$ can communicate through an undirected connected graph. In the following we suppose that $T_0 = 0$.

3. Node Sampling Service tolerant to Malicious Nodes

As mentioned in the introduction, a sampling service is a functionality local to each (correct)[1] node i, which offers to applications using it, a single primitive that returns the identifier of a random node that belongs to the system. To implement the sampling service, each node i uses all the messages it continuously receives from all the other nodes of the system. More precisely, node i uses the address (or simply the identifier) of the sender of each received message. In the following, this infinite sequence of identifiers is called the input stream of node i. For efficiency and memory constraints reasons, the sampling service must process its input stream in one pass and sequentially, and must only have access to a bounded size memory. The objective of the sampling service is to output a stream of node identifiers such that the identifier of each node of the system appears in this output stream with a probability equal to $1/n$. Note that in absence of any malicious behaviors, the sampling service could be easily implemented by relying on a reservoir sampling algorithm [30]. These algorithms randomly choose a sample of items from a sequence containing an unknown number of items, large enough so that the sequence cannot fit in memory. We denote respectively by $\sigma_i = (\sigma_i(k))_{k\geq 0}$ and by $S_i = (S_i(k))_{k\geq 0}$ the input stream and the output stream of the sampling service at any correct node i. The input stream σ_i takes its values in the set $\mathcal{N} = \{1, \ldots, n\}$. It is a sequence of independent and identically distributed (i.i.d.) random variables. Moreover, this stream is supposed to have the following property called positivity which means that each node identifier j occurs in each position k with a positive probability denoted by $p_{i,j}$. That is, for every $k \geq 0$ and for every $j \in \mathcal{N}$, we have, at correct node i,

$$p_{i,j} = \mathbb{P}\{\sigma_i(k) = j\} > 0. \qquad (H)$$

In order to simplify the notation, when node i is fixed, which is the case in what follows, we simply denote this probability by p_j, the input stream by σ and its distribution by $p = (p_1, \ldots, p_n)$. The following section describes the impact of the adversary in the constitution of σ.

3.1. Constitution of the input stream σ

Let us denote by $\mathcal{N}_m$ the subset of malicious nodes and by $\mathcal{N}_c$ the subset of correct nodes. We have $|\mathcal{N}_m| = t$ and $|\mathcal{N}_c| = n - t$. The input stream σ may be obtained by multiplexing two independent streams : an i.i.d. stream $\sigma^{(c)}$ of correct nodes with values in $\mathcal{N}_c$ and distribution $p' = (p'_j,\ j \in \mathcal{N}_c)$, $p'_j > 0$, and an i.i.d. stream of malicious nodes, produced by the adversary, with values in $\mathcal{N}_m$ and distribution $p'' =$

[1] Although malicious nodes also implement such a functionality, we cannot impose any assumptions on how they build it as their behavior can be totally arbitrary.

$(p''_j,\ j \in \mathcal{N}_m)$, $p''_j > 0$. Then the input stream $\sigma = (\sigma(k))_{k\geq 0}$ is obtained by a random multiplexing of streams $\sigma^{(c)}$ and $\sigma^{(m)}$, in the following way. We introduce a sequence $B = (B_k)_{k\geq 0}$ of i.i.d. Bernoulli random variables independent of $\sigma^{(c)}$ and $\sigma^{(m)}$, with $\mathbb{P}\{B_k = 0\} = \alpha$, where $\alpha \in (0, 1)$ is a parameter chosen by the adversary. The input stream σ is then obtained by taking successively a node id from $\sigma^{(m)}$ with probability α and from $\sigma^{(c)}$ with probability $1 - \alpha$. More formally, σ is obtained as follows.

$$\begin{aligned}\text{For all } k \geq 0, \text{ if } B_k = 0 \text{ then } \sigma(k) = \sigma^{(m)}(0) \text{ and } \forall \ell \geq 0, \sigma^{(m)}(\ell) = \sigma^{(m)}(\ell + 1)\\ \text{if } B_k = 1 \text{ then } \sigma(k) = \sigma^{(c)}(0) \text{ and } \forall \ell \geq 0, \sigma^{(c)}(\ell) = \sigma^{(c)}(\ell + 1).\end{aligned}$$

This means in practice that the stream σ is composed from $\sigma^{(c)}$ and $\sigma^{(m)}$ by taking the first entry of $\sigma^{(m)}$ with probability α and the first entry of $\sigma^{(c)}(0)$ with probability $1 - \alpha$. Note that if at $k = 0$, the first entry $\sigma^{(m)}(0)$ of $\sigma^{(m)}$ is selected then the next competition will be between $\sigma^{(c)}(0)$ and $\sigma^{(m)}(1)$, and so on. That is why we need to shift the streams in the previous procedure.

The stream σ is thus i.i.d. and its distribution $p = (p_1, \ldots, p_n)$ is given by

$$p_j = \mathbb{P}\{\sigma(k) = j\} = \begin{cases} (1 - \alpha)p'_j & \text{if} \quad j \in \mathcal{N}_c \\ \alpha p''_j & \text{if} \quad j \in \mathcal{N}_m. \end{cases}$$

The probability $\alpha \in (0, 1)$ and the number t of malicious nodes are parameters which are set by the adversary in order to bias, as much as he wishes, the distribution of the input stream σ. From a practical point of view, the adversary owns t addresses (that is t node identifiers), for any value of t smaller than n. Of course, correct nodes are not capable of distinguishing node identifiers owned by the adversary from correct ones, and do not even know t: each correct node i run its sampling algorithm fed with its input sampling σ_i, oblivious of the presence of malicious identifiers in σ_i.

3.2. The addressed problem

The goal of this paper is to design at each correct node i a Byzantine-tolerant sampling algorithm taking as input the stream σ_i and building an output stream S_i satisfying the following uniformity property.

Property 3.1 (Uniformity)**.**
For any discrete time $k \geq 0$, for any node $i \in \mathcal{N}$ and for any node $j \in \mathcal{N}$,

$$\mathbb{P}\{S_i(k) = j\} = 1/n.$$

Remark 1. Note that for each node i, the output stream S_i generated by our node sampling service does not need to be i.i.d. because it is not expected to act as the input stream of another node. As said in the introduction of the paper, a node sampling service can be used for instance in network metrology, or load balancing applications where in both cases one needs to choose uniformly at random nodes to either store or process data, or to explore or analyze the network. Our service is also an important ingredient to prevent eclipse attacks when used in a membership protocol.

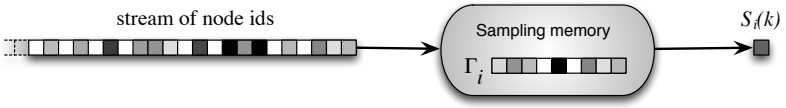


Figure 1.: Sampling component of node $i \in \mathcal{N}$.

ALGORITHM 1: Omniscient Node Sampling Algorithm run at any correct node $i \in \mathcal{N}$

Input: An arbitrary input stream σ_i;
Output: A modified output stream S_i;
Data: Γ_i a set of maximum size c. Initially, $\Gamma_i = \emptyset$;

```
1  for j ∈ σ_i do
2      if |Γ_i| < c then
3          Γ_i ← Γ_i ∪ {j};
4      else
5          with probability a_j do
6              choose ℓ from Γ_i with probability r_ℓ / Σ_{h∈Γ_i} r_h;
7              Γ_i ← (Γ_i \ {ℓ}) ∪ {j};
8          end
9      end
10     choose w from Γ_i with probability 1/|Γ_i|;
11     write w in the output stream S_i;
12 end
```

4. An Omniscient Sampling Service

This section is devoted to the design of a Byzantine-tolerant sampling service in a somehow "ideal" context, that is in a context in which it is omniscient. More precisely, we suppose that the sampling service at any correct node i knows the values of the probabilities p_j, for any $j \in \mathcal{N}$.

The omniscient sampling service has uniquely access to a set (or a data structure) Γ_i, referred to as the *sampling memory*, as illustrated in Figure 1. The maximal size of Γ_i is constant, independent from n and is denoted by c. The sampling memory contains the node ids that are selected by the algorithm when reading σ_i. Algorithm 1 describes the pseudo-code of the omniscient algorithm. This algorithm contains parameters a_j and r_j, $j \in \mathcal{N}$, that must be determined in order to obtain an output stream S_i possessing the uniformity property 3.1. Note that these parameters should be noted $a_{i,j}$ and $r_{i,j}$, but as we did for the $p_{i,j}$, we drop index i when node i is fixed.

Specifically, the omniscient algorithm at correct node i reads on the fly and sequentially the input stream and for each read element j, decides whether j is a good candidate for being stored into the constant size memory Γ_i or not. Intuitively, if p_j is very small, then j must definitely be stored into Γ_i so that j might have a chance to be part of the output stream. On the other hand, with larger p_j, there will be other opportunities for the sampler to receive j in the future. The probability to insert j in Γ_i is denoted by a_j in the algorithm. Although inserting j into Γ_i with probability a_j (the values of a_j are analyzed below) is a necessary condition to prevent very frequent ids from continuously eclipsing the ids already stored in Γ_i, this is not sufficient to guarantee that a rare id ℓ already stored in Γ_i will not be evicted each time a new id j is stored (assuming that Γ_i is full upon receipt of j). Recall that the goal of the adversary is to prevent identifiers of correct nodes to uniformly appear in the output

stream. This is achieved by removing ℓ from Γ_i with probability $r_\ell / \sum_{h \in \Gamma_i} r_h$, where $r_1, \ldots, r_n$ are positive real numbers that are analyzed below. Finally, a random node id w is chosen from Γ_i and written in the output stream (note that w is not removed from Γ_i).

In the remainder of this section, we prove that there exist both $(a_j)_{j \in \mathcal{N}}$ and $(r_j)_{j \in \mathcal{N}}$ such that the output stream provided by Algorithm 1 satisfies the uniformity property 3.1. This is achieved by modelling the receipt of node ids from σ_i by using a discrete-time homogeneous Markov chain denoted by $X^{(i)} = \{X_k^{(i)}, k \geq 0\}$. Markov chain $X^{(i)}$ represents the evolution of the node identifiers in Γ_i. Again, note that for clarity reason we shall omit the superscript i when it is clear from context. The state space S of X is defined by $S = \{A \subseteq \mathcal{N} \text{ such that } |A| = c\}$. For any $k \geq 0$, the event $X_k = A$ means that just after the k-th transition (*i.e.* the k-th received node identifier), we have $\Gamma = A$. By Algorithm 1, the transition probability matrix, denoted by P, is given for every $A, B \in S$ with $A \neq B$, by

$$P_{A,B} = \begin{cases} \dfrac{r_\ell}{r_A} p_j a_j & \text{if } A \setminus B = \{\ell\} \text{ and } B \setminus A = \{j\} \\ \\ 0 & \text{otherwise,} \end{cases}$$

where for every $A \subseteq \mathcal{N}, r_A = \sum_{h \in A} r_h$. It is easily checked that $|S| = \binom{n}{c}$. Matrix P being stochastic, for every $A \in S$,

$$\begin{aligned} P_{A,A} &= 1 - \sum_{B \in S, B \neq A} P_{A,B}. \\ &= 1 - \sum_{\ell \in A} \sum_{j \notin A} \left(\sum_{B \in S, A \setminus B = \{\ell\}, B \setminus A = \{j\}} P_{A,B} \right) \\ &= 1 - \sum_{\ell \in A} \sum_{j \notin A} \frac{r_\ell}{r_A} p_j a_j \\ &= 1 - \sum_{j \notin A} p_j a_j \\ &= 1 - \sum_{j \in \mathcal{N}} p_j a_j + \sum_{j \in A} p_j a_j. \end{aligned}$$

4.1. Stationary Analysis of Markov Chain X

The Markov chain X is clearly irreducible and aperiodic. It thus has a stationary distribution that we denote by $\pi = (\pi_A,\ A \in S)$. The row vector π is the unique solution to the linear system $\pi = \pi P$ with $\pi \mathbf{1} = 1$, where $\mathbf{1}$ is the column vector with all entries equal to 1. The symmetries observed in the transition probability matrix P gives us the intuition that X is reversible, *i.e.* that for every $A, B \in S$ we have $\pi_A P_{A,B} = \pi_B P_{B,A}$. This intuition is verified by the following theorem.

Theorem 4.1. *The Markov chain X is reversible and for every $A \in S$, we have*

$$\pi_A = \frac{r_A}{K} \left(\prod_{h \in A} \frac{p_h a_h}{r_h} \right) \tag{1}$$

where

$$K = \sum_{B \in S} r_B \left(\prod_{h \in B} \frac{p_h a_h}{r_h} \right).$$

Proof. Proof of this theorem has been previously presented in the former paper [31]. □

Let us introduce now, for every $h \in \mathcal{N}$, the subset of states $\mathcal{S}_h$ defined by

$$\mathcal{S}_h = \{A \in S \mid h \in A\}$$

and consider the probability γ_h for X to be in subset $\mathcal{S}_h$ in stationary regime. We then have

$$\gamma_h = \sum_{A \in \mathcal{S}_h} \pi_A.$$

It is easily checked, as expected, that we have

$$|\mathcal{S}_h| = \binom{n-1}{c-1} \text{ and } \sum_{h \in \mathcal{N}} \gamma_h = c.$$

4.2. Building a Uniform Node Sampler

We are now able to prove that there exist vectors $(a_j)_{j \in \mathcal{N}}$ and $(r_j)_{j \in \mathcal{N}}$ (that respectively represent the probability to insert item $j \in \mathcal{N}$ in the local memory Γ_i and the probability to remove item $j \in \mathcal{N}$ from this memory), such that the output stream S_i provided by Algorithm 1 satisfies the uniformity property 3.1. For every distribution $(p_j)_{j \in \mathcal{N}}$, we introduce the notation $q = \min_{j \in \mathcal{N}} p_j$. The probabilities p_j being positive, we have $q > 0$.

Theorem 4.2. *If, for every $j \in \mathcal{N}$ and for every $r > 0$, a_j and r_j are given by*

$$a_j = q/p_j \text{ and } r_j = r$$

then the stationary probability γ_h is given by

$$\gamma_h = c/n \text{ for all } h \in \mathcal{N}.$$

Proof. Proof of this theorem has been previously presented in the former paper [31]. □

4.3. Transient Analysis of Markov Chain X

This section is devoted to the analysis of the transient behavior of the Markov chain X. For every $k \geq 0$, we denote by $\pi(k) = (\pi_A(k),\ A \in S)$ the row vector containing the distribution of X at instant k, which is defined, for $k \geq 0$ and $A \in S$, by

$$\pi_A(k) = \mathbb{P}\{X_k = A\}.$$

Vector $\pi(k)$ is given by

$$\pi(k) = \pi(0)P^k,$$

where $\pi(0)$ is the initial distribution of X. For every $h \in \mathcal{N}$, we denote by $\gamma_h(k)$ the probability for X to be in subset $\mathcal{S}_h$ at instant k. This probability is given by

$$\gamma_h(k) = \mathbb{P}\{X_k \in \mathcal{S}_h\} = \sum_{A \in \mathcal{S}_h} \pi_A(k) = \sum_{A \in \mathcal{S}_h} \left(\pi(0)P^k\right)_A.$$

The integer h being fixed, we consider the partition $\mathcal{S}_h$, $\mathcal{S}'_h$ of the state space S, where $\mathcal{S}'_h = S \setminus \mathcal{S}_h$. We decompose the transition probability matrix P with respect to that partition by writing

$$P = \begin{pmatrix} P_{\mathcal{S}_h} & P_{\mathcal{S}_h\mathcal{S}'_h} \\ P_{\mathcal{S}'_h\mathcal{S}_h} & P_{\mathcal{S}'_h} \end{pmatrix},$$

where matrix $P_{\mathcal{S}_h}$ (resp. $P_{\mathcal{S}_h\mathcal{S}'_h}$) is the submatrix of P containing the transition probabilities from states of $\mathcal{S}_h$ to states of $\mathcal{S}_h$ (resp. $\mathcal{S}'_h$) and $P_{\mathcal{S}'_h}$ (resp. $P_{\mathcal{S}'_h\mathcal{S}_h}$) is the submatrix of P containing the transition probabilities from states of $\mathcal{S}'_h$ to states of $\mathcal{S}'_h$ (resp. $\mathcal{S}_h$). We also introduce the $|S|$-dimensional column vector $\mathbf{1}_h$ defined by $\mathbf{1}_h(A) = 1_{\{h \in A\}}$. With respect to the partition $\mathcal{S}_h$, $\mathcal{S}'_h$, vector $\mathbf{1}_h$ writes $\mathbf{1}_h = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix}$. We recall that vector $\mathbf{1}$ (resp. $\mathbf{0}$) is a column vector with all its entries equal to 1 (resp. 0), its dimension being given by the context of its use. Here the dimension of column vector $\mathbf{1}$ (resp. $\mathbf{0}$) is $|\mathcal{S}_h|$ (resp. $|\mathcal{S}'_h|$). Using this partition and these notations, we get, for every $k \geq 0$,

$$\gamma_h(k) = \pi(0)P^k\mathbf{1}_h.$$

Note that, for every $A \in \mathcal{S}_h$, we have

$$\begin{aligned}
(P_{\mathcal{S}_h}\mathbf{1})_A &= P_{A,A} + \sum_{B\in\mathcal{S}_h, B\neq A} P_{A,B} \qquad (2)\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{\substack{j\in A \text{ and } \ell\in B\\ \text{s.t. } B\in\mathcal{S}_h, B\neq A}} \frac{r_\ell p_j a_j}{r_A} 1_{\{A\setminus B=\ell, B\setminus A=\{j\}\}}\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{\ell\in A\setminus\{h\}} \sum_{j\in\mathcal{N}\setminus A} \frac{r_\ell}{r_A} p_j a_j\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \left(1 - \frac{r_h}{r_A}\right) \sum_{j\in\mathcal{N}\setminus A} p_j a_j\\
&= 1 - \frac{r_h}{r_A} \sum_{j\in\mathcal{N}\setminus A} p_j a_j. \qquad (3)
\end{aligned}$$

The matrix P being stochastic, we get

$$\left(P_{\mathcal{S}_h\mathcal{S}'_h}\mathbf{1}\right)_A = \frac{r_h}{r_A} \sum_{j\in\mathcal{N}\setminus A} p_j a_j.$$

In the same way, we have, for every $A \in \mathcal{S}'_h$,

$$\begin{aligned}
\left(P_{\mathcal{S}'_h}\mathbf{1}\right)_A &= P_{A,A} + \sum_{B\in\mathcal{S}'_h, B\neq A} P_{A,B}\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{\substack{j\in A \text{ and } \ell\in B\\ \text{s.t. } B\in\mathcal{S}'_h, B\neq A}} \frac{r_\ell p_j a_j}{r_A} 1_{\{A\setminus B=\ell, B\setminus A=\{j\}\}}\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{\ell\in A} \sum_{j\in(\mathcal{N}\setminus A)\setminus\{h\}} \frac{r_\ell}{r_A} p_j a_j\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{j\in(\mathcal{N}\setminus A)\setminus\{h\}} p_j a_j\\
&= 1 - \sum_{j\in\mathcal{N}} p_j a_j + \sum_{j\in A} p_j a_j + \sum_{j\in\mathcal{N}\setminus A} p_j a_j - p_h a_h\\
&= 1 - p_h a_h.
\end{aligned}$$

Note that this quantity is independent of $A \in \mathcal{S}'_h$. This means that we have

$$P_{\mathcal{S}'_h}\mathbf{1} = (1 - p_h a_h)\mathbf{1}.$$

The matrix P being stochastic, we get

$$P_{\mathcal{S}'_h\mathcal{S}_h}\mathbf{1} = p_h a_h \mathbf{1}.$$

We are now able to state the following theorem.

Theorem 4.3. *If, for every $j \in \mathcal{N}$ and for every $r > 0$, a_j and r_j are given by*

$$a_j = q/p_j \text{ and } r_j = r$$

then the transient probability $\gamma_h(k)$ is given, for every $h \in \mathcal{N}$ and $k \geq 0$, by

$$\gamma_h(k) = \frac{c}{n} + \left(1 - \frac{qn}{c}\right)^k \left(\pi_{\mathcal{S}_h}(0) - \frac{c}{n}\right),$$

where $\pi_{\mathcal{S}_h}(0)$ is the probability for Markov chain X to start in subset $\mathcal{S}_h$.

Proof. We use Relation (2). Since $a_j = q/p_j$ and $r_j = r$, we have, for every $h \in \mathcal{N}$ and $A \in \mathcal{S}_h$ and recalling that $|A| = c$,

$$(P_{\mathcal{S}_h}\mathbf{1})_A = 1 - q\frac{n-c}{c} \quad \text{and} \quad \left(P_{\mathcal{S}_h\mathcal{S}'_h}\mathbf{1}\right)_A = q\frac{n-c}{c}.$$

This rewrites as

$$P_{\mathcal{S}_h}\mathbf{1} = \left(1 - q\frac{n-c}{c}\right)\mathbf{1} \quad \text{and} \quad P_{\mathcal{S}_h\mathcal{S}'_h}\mathbf{1} = q\frac{n-c}{c}\mathbf{1}. \tag{4}$$

In the same way, we get

$$P_{\mathcal{S}'_h}\mathbf{1} = (1-q)\,\mathbf{1} \quad \text{and} \quad P_{\mathcal{S}'_h\mathcal{S}_h}\mathbf{1} = q\mathbf{1}. \tag{5}$$

For every $h \in \mathcal{N}$, we introduce the discrete-time stochastic process $Y^{(h)} = \{Y_k^{(h)},\ k \geq 0\}$ over the state space $\{s_h, s'_h\}$ defined, for every $k \geq 0$, by

$$Y_k^{(h)} = s_h \text{ (resp. } s'_h) \Longleftrightarrow X_k \in \mathcal{S}_h \text{ (resp. } \mathcal{S}'_h).$$

From Relations (4) and (5), it is easily checked using the results on state aggregation in Markov chains, (see [32] or [33]), that $Y^{(h)}$ is a homogeneous Markov chain with transition probability matrix Q given by

$$Q = \begin{pmatrix} 1 - q\dfrac{n-c}{c} & q\dfrac{n-c}{c} \\ \\ q & 1-q \end{pmatrix}.$$

If the initial distribution $\pi(0)$ of X is decomposed through the partition $\mathcal{S}_h$, $\mathcal{S}'_h$ of S as

$$\pi(0) = \left(\pi^{(\mathcal{S}_h)}(0), \pi^{(\mathcal{S}'_h)}(0)\right),$$

then the initial distribution $\alpha = \left(\alpha_{s_h}, \alpha_{s'_h}\right)$ of Y is given by

$$\alpha_{s_h} = \pi^{(\mathcal{S}_h)}(0)\mathbf{1} \quad \text{and} \quad \alpha_{s'_h} = \pi^{(\mathcal{S}'_h)}(0)\mathbf{1} = 1 - \alpha_{s_h}.$$

These results lead, for every $h \in \mathcal{N}$ and $k \geq 0$, to

$$\gamma_h(k) = \pi(0)P^k \mathbf{1}_h = \alpha Q^k e_1,$$

where e_1 is the 2-dimensional column vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$. It is then very easy to check that we have, for every $h \in \mathcal{N}$ and $k \geq 0$,

$$\gamma_h(k) \quad = \quad \frac{c}{n} + \left(1 - \frac{qn}{c}\right)^k \left(\alpha_{s_h} - \frac{c}{n}\right)$$

which completes the proof. □

The following corollary summarizes the analysis.

Corollary 4.4. *Given any arbitrary i.i.d. input stream satisfying the positivity property (H), Algorithm 1 outputs a stream that satisfies the uniformity property 3.1 if*

$$\forall j \in \mathcal{N}, a_j = \frac{q}{p_j} \text{ and } r_j = r, \text{ for every } r > 0.$$

Proof. Let any correct node i run Algorithm 1. By assumption, at any discrete time k, every node j in $\mathcal{N}$ has a probability $p_j > 0$ to feed Algorithm 1. Thus, $a_j > 0$ for every $j \in \mathcal{N}$. From Theorem 4.2, when k tends to infinity, any node j has a probability $\gamma_j = c/n$ to be in the sampler memory. From Algorithm 1, the output of the sampler is any node from Γ_i chosen with probability $1/c$. Thus j appears as an output with probability $1/n$, which ensures the uniformity property 3.1. □

5. A Knowledge-free Sampling Service

We have proposed in Section 4 a local algorithm capable of building a uniform stream on the fly, from any arbitrary i.i.d. input stream σ satisfying the positivity property (H). This local algorithm uses a constant amount of memory, and does not need to know ahead of time which node identifiers will appear in σ. However it needs to know, upon receipt of a data item j, its probability of occurrence p_j in σ (which is the reason why we call it omniscient). Clearly such an assumption is unrealistic in dynamic systems and moreover, the adversary may modify the occurrence probability of any node identifier in the stream by increasing the number of occurrences of the t node identifiers it manipulates.

In this section, we propose an algorithm, called hereafter *knowledge-free algorithm*, that does not assume that probabilities $p_1, \ldots, p_n$ are known. For each received node id j from σ, the proposed algorithm selects the identifier that will be part of the output stream by solely relying on an estimation of the frequency f_j of node j. Such an estimation is computed on the fly by using a sublinear number of bits in the population size of the system. Specifically, the knowledge-free algorithm uses one additional data structure with respect to the omniscient one, as illustrated in Figure 2. This data structure is the *Count-Min Sketch* $\hat{F}$ proposed by Cormode and Muthukrishnan [34]. Sketch $\hat{F}$ is built on the fly and provides at any time, and for any j read from σ, an approximation $\hat{f}_j$ of the number f_j of times j has appeared in σ from the inception of the stream. For self-containment reasons, we briefly describe how $\hat{F}$ is built.

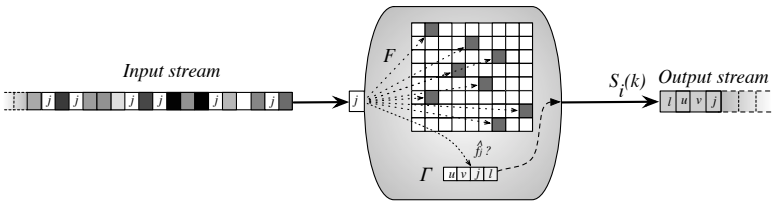


Figure 2.: Sampling Sketch of node $i \in \mathcal{N}$.

5.1. Frequency Estimation of each Item in the Stream

For any item j in the input stream σ, the algorithm proposed by Cormode and Muthukrishnan [34] outputs an estimation $\hat{f}_j$ of the number f_j of times j has occurred in the stream so far. The error of the estimator in answering a query for $\hat{f}_j$ is within a factor of $\varepsilon(m - f_j)$ with probability δ, where m represents the number of items read from the input stream. The estimation is computed by maintaining a two-dimensional array $\hat{F}$ of $s_1 s_2$ counters with $s_1 = \lceil \log(1/\delta) \rceil$ and $s_2 = \lceil e/\varepsilon \rceil$ (where $e = \exp(1)$), and by using 2-universal hash functions $h_1, \ldots, h_{s_1}$.

A collection H of hash functions $h : \{1, \ldots, M\} \to \{1, \ldots, M'\}$ is said to be *2-universal* if for every two different items $x, y \in \{1, \ldots, M\}$,

$$\forall h \in H, \mathbb{P}\{h(x) = h(y)\} \leq \frac{1}{M'},$$

which is exactly the probability of collision obtained if the hash function assigns truly random values to any $x \in \{1, \ldots, M\}$.

Each time an item j is read from the input stream, this causes one counter per line to be incremented, *i.e.* $\hat{F}[v][h_v(j)]$ is incremented for all $v \in \{1, \ldots, s_1\}$. Thus at any time, the sum of the buckets of any given line is equal to the number of items read from the input stream. When a query is issued to get an estimate $\hat{f}_j$ of the frequency of j (*i.e.* the number of occurrences of j read so far from the stream), the returned value corresponds to the minimum among the s_1 values of $\hat{F}[v][h_v(j)]$, $v \in \{1, \ldots, s_1\}$. Space required by this algorithm is proportional to $\log(1/\delta)/\varepsilon$, and the update time per element is significantly sublinear in the size of the sketch [34], which makes this algorithm fully adapted to our context. Specifically, authors of [34] have shown that, after reading m items from the stream, we have

$$\forall j \in \mathcal{N},\ \mathbb{P}\left\{\hat{f}_j - f_j \geq \varepsilon\,(m - f_j)\right\} \leq \delta. \tag{6}$$

5.2. The Knowledge-free Sampling Algorithm

The knowledge-free data-stream algorithm we propose is a simple extension of the omniscient one, where the insertion probability a_j for any received $j \in \sigma$ is computed by using the estimation $\hat{f}_j$ provided by Algorithm [34]. The pseudo-code of our algorithm is presented in Algorithm 2. Note that the instructions **cobegin** at lines 2 and 5 mean that codes of Algorithm [34] and lines (6–17) are executed in parallel (at any discrete time k the first id of σ is read by both codes). We analyze in Section 6 the resilience of Algorithm 2 against representative attacks, *i.e.* targeted and flooding attacks. Then in Section 7 we evaluate with extensive simulations the quality of the knowledge-free

algorithm with respect to the omniscient one.

ALGORITHM 2: Knowledge-free data-stream algorithm run at any correct node $i \in \mathcal{N}$

Input: An arbitrary input stream σ_i; real values δ and ε;
Output: An (ε, δ) uniform output stream S_i;
Data: Γ_i a set of maximum size c. Initially, $\Gamma_i = \emptyset$;
Data: $\hat{F}$ the Count-min Sketch matrix

```
1  for j ∈ σ_i do
2      cobegin
3          execute Algorithm [34] using ε and δ/2 as parameters;
4      end
5      cobegin
6          f̂_j ← Estimate(f_j);
7          q̂ ← min_{1≤u≤s1, 1≤v≤s2} F̂[u][v];
8          if |Γ_i| < c then
9              Γ_i ← Γ_i ∪ {j};
10         else
11             with probability â_j = q̂/f̂_j do
12                 choose ℓ from Γ_i with probability 1/c;
13                 Γ_i ← (Γ_i \ {ℓ}) ∪ {j};
14             end
15         end
16         choose w from Γ_i with probability 1/|Γ_i|;
17         write w in the output stream S_i;
18     end
19 end
```

5.3. Complexity Analysis

In this section, we show that the knowledge-free data-stream algorithm provides an efficient approximation of the omniscient one, without requiring any a priori knowledge neither on the size of the input stream, nor on the number of distinct elements that compose it, nor on the probability distribution of these elements. For every $\ell \in \mathcal{N}$, we denote by $\hat{a}_\ell$ (respectively $\hat{p}_\ell$), the estimation of a_ℓ (respectively p_ℓ) defined in Section 4. Finally, and as above, m represents the current size of the input stream (*i.e.* the number of node ids that have been read so far from the input stream).

Theorem 5.1. *For every $\ell \in \mathcal{N}$, for every m, the knowledge-free algorithm implemented in Algorithm 2 satisfies*

$$\mathbb{P}\left\{|\hat{a}_\ell - a_\ell| \geq \varepsilon \left(\frac{1}{p_\ell} - a_\ell\right)\right\} \leq \delta$$

where $\varepsilon > 0$ and $\delta \in (0, 1)$ are two real values introduced in Section 5.1, and p_ℓ is the empirical probability of occurrence of item ℓ in the input stream, i.e. $p_\ell = f_\ell/m$.

Proof. Since $p_j = f_j/m$ and $\hat{p}_j = \hat{f}_j/m$, we get from [34], for any $j \in \mathcal{N}$,

$$\mathbb{P}\left\{\hat{p}_j - p_j \geq \varepsilon\,(1 - p_j)\right\} = \mathbb{P}\left\{\hat{f}_j - f_j \geq \varepsilon\,(m - f_j)\right\} \leq \frac{\delta}{2}.$$

We introduce the notation

$$A = \mathbb{P}\left\{\hat{a}_\ell - a_\ell \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \tag{7}$$

and

$$B = \mathbb{P}\left\{\hat{a}_\ell - a_\ell \leq -\varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \tag{8}$$

We first focus on Relation (7). For every $u \in \{1, \ldots, s_1\}$ and $\ell \in \mathcal{N}$, we denote by $W_{u,\ell}$ the random variable defined by

$$W_{u,\ell} = \frac{\min_{1\leq j\leq s_2} \hat{F}[u][j]}{\min_{1\leq v\leq s_1} \hat{F}[v][h_v(\ell)]} - a_\ell,$$

where a_ℓ is given in Corollary 4.4. We then have

$$\mathbb{E}[W_{u,\ell}] = \mathbb{E}\left[\frac{\min_{1\leq j\leq s_2} \hat{F}[u][j]}{\min_{1\leq v\leq s_1} \hat{F}[v][h_v(\ell)]}\right] - a_\ell.$$

From [34], we have $s_2 = \lceil e/\varepsilon \rceil$. We first prove that, for any $u \in \{1, \ldots, s_1\}$, the following statement holds:

$$\min_{1\leq j\leq s_2} \hat{F}[u][j] \leq \frac{m}{s_2}. \tag{9}$$

Suppose by contradiction that $\min_{1\leq j\leq s_2} \hat{F}[u][j] > m/s_2$. Thus, $\sum_{1\leq j\leq s_2} \hat{F}[u][j] > m$ which is impossible by construction. From Relation (9), and since $\min_{1\leq v\leq s_1} \hat{F}[v][h_v(\ell)] \geq mp_\ell$ [34], we then have

$$\mathbb{E}[W_{u,\ell}] \leq \frac{1}{s_2 p_\ell} - a_\ell \leq \frac{1}{s_2}\left(\frac{1}{p_\ell} - a_\ell\right).$$

Using the Markov inequality, and $s_2 = \lceil e/\varepsilon \rceil$, we then get

$$\begin{aligned}
\mathbb{P}\left[W_{u,\ell} \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right] &\leq \mathbb{P}\left[\, W_{u,\ell} \geq \varepsilon s_2 \mathbb{E}[W_{u,\ell}] \,\right] \\
&\leq \frac{\mathbb{E}[W_{u,\ell}]}{\varepsilon s_2 \mathbb{E}[W_{u,\ell}]} = \frac{1}{\varepsilon s_2} \\
&\leq \frac{1}{2}.
\end{aligned}$$

The above probability holds for any line u of $\hat{F}$. From both Algorithms [34] and 2, we

have

$$\begin{aligned}\hat{a}_\ell - a_\ell &= \frac{\min_{1\leq u\leq s_1, 1\leq j\leq s_2} \hat{F}[u][j]}{\min_{1\leq v\leq s_1} \hat{F}[v][h_v(\ell)]} - a_\ell \\ &= \min_{1\leq u\leq s_1} \left(\frac{\min_{1\leq j\leq s_2} \hat{F}[u][j]}{\min_{1\leq v\leq s_1]} \hat{F}[v][h_v(\ell)]} - a_\ell \right) \\ &= \min_{1\leq u\leq s_1} W_{u,\ell}\end{aligned}$$

By construction of the pairwise hash functions $h_1, \ldots, h_{s_1}$, array $\hat{F}$ is made of s_1 independent lines. Thus, we obtain

$$\begin{aligned}\mathbb{P}\left\{\hat{a}_\ell - a_\ell \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} &= \mathbb{P}\left\{\min_{1\leq u\leq s_1} W_{u,\ell} \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \\ &= \mathbb{P}\left\{W_{1,\ell} \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right), \ldots, W_{s_1,\ell} \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \\ &= \prod_{u\in[s_1]} \mathbb{P}\left\{W_{u,\ell} \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \\ &\leq \frac{1}{2^{s_1}} \leq \frac{\delta}{2}. \qquad (10)\end{aligned}$$

We now focus on Relation (8). By extension of the notation $q = \min_{j\in\mathcal{N}} p_j$ used in Section 4.2, we define $\hat{q} = \min_{j\in\mathcal{N}} \hat{p}_j$. We then have

$$\begin{aligned}\mathbb{P}\left\{\hat{a}_\ell - a_\ell \leq -\varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} &= \mathbb{P}\left\{\frac{\hat{q}}{\hat{p}_\ell} - \frac{q}{p_\ell} \leq -\varepsilon\left(\frac{1}{p_\ell} - \frac{q}{p_\ell}\right)\right\} \\ &= \mathbb{P}\left\{\frac{\hat{p}_\ell q - p_\ell \hat{q}}{\hat{p}_\ell p_\ell} \geq \varepsilon\left(\frac{1-q}{p_\ell}\right)\right\}.\end{aligned}$$

By definition $\hat{p}_\ell \geq \hat{q}$ and by Count-Min sketch, $\hat{q} \geq q$. Thus,

$$\begin{aligned}\mathbb{P}\left\{\hat{a}_\ell - a_\ell \leq -\varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} &\leq \mathbb{P}\left\{\left(\hat{p}_\ell - p_\ell\right)\hat{q} \geq \varepsilon\hat{p}_\ell\left(1-q\right)\right\} \\ &\leq \mathbb{P}\left\{\left(\hat{p}_\ell - p_\ell\right)\hat{p}_\ell \geq \varepsilon\hat{p}_\ell\left(1-q\right)\right\} \\ &= \mathbb{P}\left\{\hat{p}_\ell - p_\ell \geq \varepsilon\left(1-q\right)\right\}.\end{aligned}$$

Finally, from Relation (6) and since $1 - q \geq 1 - p_\ell$, we have

$$\begin{aligned}\mathbb{P}\left\{\hat{a}_\ell - a_\ell \leq -\varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} &\leq \mathbb{P}\left\{\hat{p}_\ell - p_\ell \geq \varepsilon\left(1-p_\ell\right)\right\} \\ &\leq \frac{\delta}{2}. \qquad (11)\end{aligned}$$

From Relations (10) and (11), we obtain

$$\mathbb{P}\left\{|\hat{a}_\ell - a_\ell| \geq \varepsilon\left(\frac{1}{p_\ell} - a_\ell\right)\right\} \leq \delta, \qquad (12)$$

which completes the proof. □

This completes the time and space analysis of Algorithm 2 which is summarized in the following Corollary.

Corollary 5.2. *The knowledge-free algorithm whose pseudo-code is presented in Algorithm 2 uses $\mathcal{O}(\log(n)\log(1/\delta)/\varepsilon)$ bits of space to approximate a uniform output stream from an arbitrarily biased input one.*

Proof. By Theorem 5.1, Algorithm 2 returns an approximation of $(p_\ell)_{\ell\in\mathcal{N}}$ and $(a_\ell)_{\ell\in\mathcal{N}}$ used in the omniscient algorithm presented in Algorithm 1. By Corollary 4.4, the omniscient algorithm implements a node sampling service robust to any biased input stream. Thus the second part of the corollary holds. Now $\log(1/\delta)\log(n)/\varepsilon$ bits of space are needed to approximate node id frequencies, and $c\log(n)$ bits of space are needed by the sampling memory Γ. By construction, c is constant, which concludes the proof. □

6. The reason why our Algorithm is Tolerant to Collusions of Malicious Nodes

As said in Section 2.2, we suppose that the adversary has enough resources to generate a large number t of node identifiers in the input stream σ_i of any correct node i in order to prevent the sampler service of i to output a uniform stream S_i. In this section, we derive the minimum number of identifiers the adversary has to generate to subvert the node sampling service.

We have shown in Section 4 that the omniscient algorithm is capable of building a uniform stream from any arbitrary i.i.d. input stream. We have shown in Section 5 that the knowledge-free algorithm, implemented by Algorithm 2, is an (ε, δ)-approximation of the omniscient algorithm. Thus the only latitude given to the adversary to bias the output stream of any correct node is to increase the error made on the estimations $\hat{f}_j$ with $j \in \mathcal{N}$. By construction of the Coun-Min sketch Algorithm [34], each received element j is mapped to exactly one entry in each row of matrix $\hat{F}$, and each of these entries is incremented by one. Thus to disrupt the estimation of any $\hat{f}_j$, the adversary has to generate sufficiently many node identifiers $o_1, \ldots, o_t$ such that for all $v \in \{1, \ldots, s_1\}$, there exists $i \in \{1, \ldots, t\}$ such that $h_v(o_i) = h_v(j)$. By doing so the estimation $\hat{f}_j$ will be arbitrarily overestimated, and thus, by Algorithm 2, j will occur in the output stream with an arbitrary smaller frequency. We call this attack a *targeted attack*. Note that, the adversary will blindly bias the frequency estimation of several node identifiers, including its owns, *i.e.*, $o_1, \ldots, o_t$. A flooding attack consists for the adversary in overestimating all the node identifiers. We now analyze the minimum effort that needs to be exerted by the adversary to make a targeted attack successful with probability $1 - \eta_T$ where $\eta_T < 1$.

6.1. Analysis of the Effort Needed to Make a Targeted Attack Successful

We model a targeted attack as an urn problem, where each entry of $\hat{F}$ is modeled as an urn and each received distinct node identifier as a ball. Consider a set of s_2 urns initially empty in which we throw balls, one by one, according to the uniform distribution (by definition of 2-universal hash functions, each ball has an equal probability to be thrown

in any of the s_2 urns, see Section 2). We denote by N_k the number of non empty urns at time k, *i.e.*, just after the throwing of the k-th ball and we consider the integer L_{s_2}, which counts the number of balls needed to get a collision with a probability greater than $1 - \eta_T$. Formally, for a given value of s_2 and $\eta_T \in (0, 1)$, we have

$$L_{s_2} = \inf\{k \geq 2 \mid \mathbb{P}\{N_k = N_{k-1}\} > 1 - \eta_T\}.$$

In the knowledge-free algorithm, the previous experiment is executed identically and independently in s_1 sets of s_2 urns. At each time, we throw in parallel s_1 balls, one in each set of s_2 urns. For $i = 1, \ldots, s_1$, the random variable $N_k^{(i)}$ counts the number of non empty urns among the i-th set of s_2 urns at time k and we consider the integer L_{s_2,s_1} which counts the number of balls needed to get a collision in each set of s_2 urns, with a probability greater than $1 - \eta_T$. We thus have in particular $L_{s_2} = L_{s_2,1}$. More formally, for given values of s_1, s_2 and $\eta_T \in (0, 1)$, integer L_{s_2,s_1} is defined by

$$L_{s_2,s_1} = \inf\{k \geq 2 \mid \mathbb{P}\{N_k^{(1)} = N_{k-1}^{(1)}, \ldots, N_k^{(s_1)} = N_{k-1}^{(s_1)}\} > 1 - \eta_T\}.$$

Since the s_1 experiments in parallel are identical and independent, the random variables $N_k^{(1)}, \ldots, N_k^{(s_1)}$ are, for each $k \geq 1$, independent and identically distributed. It is thus sufficient to consider a single set of s_2 urns and L_{s_2,s_1} is then given by

$$L_{s_2,s_1} = \inf\{k \geq 2 \mid (\mathbb{P}\{N_k = N_{k-1}\})^{s_1} > 1 - \eta_T\}. \tag{13}$$

The random variable N_k takes its values in the set $\{1, \ldots, s_2 \wedge k\}$, where $s_2 \wedge k = \min\{s_2, k\}$. The distribution of N_k is given, for every $s_2 \geq 1$ and $k \geq 1$, by the following theorem which uses the Stirling numbers $S(k, i)$ of the second kind. These numbers are defined, for $k \geq 1$ and $i = 1, \ldots, k$, by the relations $S(1, 1) = 1$ and

$$S(k, i) = S(k - 1, i - 1)1_{\{i \neq 1\}} + iS(k - 1, i)1_{\{i \neq k\}}. \tag{14}$$

It is well-known that this recursion leads to the explicit formula

$$S(k, i) = \frac{1}{i!} \sum_{h=0}^{i} (-1)^h \binom{i}{h} (i - h)^k. \tag{15}$$

Theorem 6.1. *For every $s_2 \geq 1$, $k \geq 1$ and $i = 1, \ldots, s_2 \wedge k$, we have*

$$\mathbb{P}\{N_k = i\} = \frac{S(k, i) s_2!}{{s_2}^k (s_2 - i)!}.$$

Proof of this theorem has been previously presented in the former paper [31].

We are now able to compute, for every $k \geq 2$, the probabilities $\mathbb{P}\{N_k = N_{k-1}\}$. We

have

$$\begin{aligned}\mathbb{P}\{N_k = N_{k-1}\} &= \sum_{i=1}^{s_2 \wedge (k-1)} \mathbb{P}\{N_k = i \mid N_{k-1} = i\}\mathbb{P}\{N_{k-1} = i\} \\ &= \frac{1}{s_2} \sum_{i=1}^{s_2 \wedge (k-1)} i\mathbb{P}\{N_{k-1} = i\} = \frac{\mathbb{E}(N_{k-1})}{s_2}\end{aligned}$$

The number of balls L_{s_2,s_1} needed to get a collision in each set of s_2 urns is linear in s_2 and sublinear in s_1 and η_T which explains why attacking a single node requires a significant number of distinct malicious node identifiers. For instance, when $s_2 = 50$ and $s_1 = 10$, the adversary has to inject in the input stream 150 distinct node identifiers to have no more than 50% of chance to get its targeted attack successful. On the other hand, with the same settings of s_2 and s_1, 571 distinct node identifiers need to be injected to guarantee with probability 0.9999 a successful targeted attack. Note that this analysis, as well as the one presented in Section 6.2, derives the minimum number of distinct identifiers that need to be injected by the adversary in a given stream σ to bias the output stream. It does not consider the frequency at which these identifiers must appear in the input stream σ to significantly impact the estimation of the identifiers that share the same entry in the sketch. As said in Section 2, the effort required by an adversary to bias the output stream is not in the repeated injection of node identifiers in σ but rather on the cost of creation of these identifiers. Indeed, to own an identifier, a node typically needs to interact with a central authority to receive a certificate assessing the validity and integrity of the identifier. The impact at which node identifiers recur in the input stream is analyzed in Section 7.

6.2. Analysis of the Effort Needed to Make a Flooding Attack Successful

We now analyze the minimum effort that needs to be exerted by the adversary to make a flooding attack successful with probability $1 - \eta_F$ where $\eta_F < 1$. As for the targeted attack, we model this attack as an urn problem, where as previously, each entry is modeled as an urn and each distinct node identifier received as a ball. Let U_{s_2} be the number of balls needed in order to obtain all the s_2 urns occupied, *i.e.* with at least one ball. It is easily checked that $\mathbb{P}\{U_1 = 1\} = 1$ and that, for $k \geq s_2 \geq 2$, we have

$$U_{s_2} = k \Longrightarrow N_{k-1} = s_2 - 1.$$

We thus have

$$\begin{aligned}\mathbb{P}\{U_{s_2} = k\} &= \mathbb{P}\{U_{s_2} = k, N_{k-1} = s_2 - 1\} \\ &= \mathbb{P}\{U_{s_2} = k \mid N_{k-1} = s_2 - 1\}\mathbb{P}\{N_{k-1} = s_2 - 1\} \\ &= \frac{1}{s_2}\mathbb{P}\{N_{k-1} = s_2 - 1\}.\end{aligned}$$

From Theorem 6.1 and Relation (15), we get, for $s_2 \geq 2$ and $k \geq s_2$,

$$\mathbb{P}\{U_{s_2} = k\} = \frac{S(k-1, s_2-1)(s_2-1)!}{{s_2}^{k-1}} = \frac{1}{{s_2}^{k-1}} \sum_{r=0}^{s_2-1} (-1)^r \binom{s_2-1}{r} (s_2-1-r)^{k-1}.$$

Finally, we consider the integer E_{s_2} which counts the number of balls needed to get a collision in all the $s_1 s_2$ urns. Note that this number is independent of s_1 as by definition, the s_1 experiments in parallel are identical and independent. Thus, filling entirely a set of s_2 urns leads to obtain all the s_1 sets of s_2 urns occupied. For given value of s_2 and $\eta_F \in (0,1)$, integer E_{s_2} is defined by

$$E_{s_2} = \inf \left\{ k \geq s_2 \left| \sum_{i=s_2}^{k} \mathbb{P}\{U_{s_2} = i\} > 1 - \eta_F \right. \right\}. \tag{16}$$

Table 1.: Key values of L_{s_2,s_1} and E_{s_2}.

Settings		η_T or η_F	L_{s_2,s_1}	E_{s_2}
Error ($s_2 = \lceil e/\varepsilon \rceil$)	Precision ($s_1 = \lceil \log(1/\delta) \rceil$)			
10 ($\varepsilon \sim 0.3$)	5 ($\delta \sim 10^{-2}$)	10^{-1}	38	44
10	5	10^{-4}	104	110
15	10	10^{-1}	68	73
15	10	10^{-4}	168	173
50 ($\varepsilon \sim 0.05$)	5	10^{-1}	193	306
50	10 ($\delta \sim 10^{-3}$)	10^{-1}	227	
50	40 ($\delta \sim 10^{-12}$)	10^{-1}	296	
50	5	10^{-4}	537	651
50	10	10^{-4}	571	
50	40	10^{-4}	640	
250 ($\varepsilon \sim 0.01$)	10	10^{-1}	1,138	1,617
250	10	10^{-4}	2,871	3,363

The number E_{s_2} of distinct ids the adversary has to inject in the input stream to introduce a bias on the identifiers of all the correct nodes actually shows the upper bound of L_{s_2,s_1} given s_2 and $\eta_T = \eta_F$. Making a flooding attack successful with probability 0.9 when $s_2 = 50$ requires around 300 malicious identifiers, while it requires around 650 node identifiers when the desired probability of success is equal to 0.9999.

The main results of both analyses are summarized in Table 1. The most important one is that the effort that needs to be exerted by the adversary to subvert the sampling service can be made arbitrarily large by any correct node by just increasing the memory space of the sampler. The second one, which derives from the first one, is the absence of relationship between the effort of the adversary and the size of the population size, which guarantees the scalability of our node sampler service.

7. Performance Evaluation of the Node Sampling Service

7.1. Settings of the Experiments

We have implemented both the omniscient and knowledge-free algorithms of the node sampling service and have conducted a series of experiments on different types of streams and for different parameters settings. We have fed our algorithms with both real-world data sets and synthetic traces. Real data give a realistic representation of some existing systems, while the latter ones allow us to capture phenomenon which may be difficult to obtain from real-world traces, and thus allow us to check the robustness of our algorithms. We have varied all the significant parameters of our algorithm, that is, the size m of the stream, the number of distinct data items n in each stream, the size of the local memory c, the number s_2 of entries in each line of the Count-Min Sketch matrix, and the number s_1 of lines of this matrix. For each parameters setting, we have conducted and averaged 100 trials of the same experiment, leading to a total of more than $100,000$ experiments for the evaluation of our algorithms. Real data have been downloaded from the repository of Internet network traffic [35]. We have used three large traces among the available ones. The first one represents one month of HTTP requests to the NASA Kennedy Space Center WWW server, the second one contains two weeks logs of HTTP requests to the Internet service provider ClarkNet WWW server (ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area), and the last one represents seven months of HTTP requests to the WWW server of the University of Saskatchewan, Canada. These data sets will be respectively referred to as NASA, ClarkNet, and Saskatchewan traces in the remaining of the paper. Table 2 presents some statistics of these data traces, in term of stream size (*cf.* "# ids"), population size in each stream (*cf.* "# distinct ids") and the number of occurrences of the most frequent id (*cf.* "max. freq."). Note that all these benchmarks share a Zipfian behaviour, with a lower α parameter for the University of Saskatchewan.

Table 2.: Statistics of real data traces

Data trace	# ids (m)	# distinct ids (n)	max. freq.
NASA	1,891,715	81,983	17,572
ClarkNet	1,673,794	94,787	7,239
Saskatchewan	2,408,625	162,523	52,695

7.2. Main Lessons drawn from the Experiments

We now present the main lessons drawn from these experiments. As said above, these experiments aim at showing the impact of over-represented (malicious) node identifiers in the input stream of the sampler service.

In order to evaluate the accuracy of our algorithm at node i, we measure the Euclidean distance between the output stream S_i and the uniform one, denoted by $\mathcal{D}(S_i,\mathcal{U})$. Note that all the distance measures in the Ali-Silvey distances are applicable to quantifying statistical differences between data streams. Based on the Euclidean

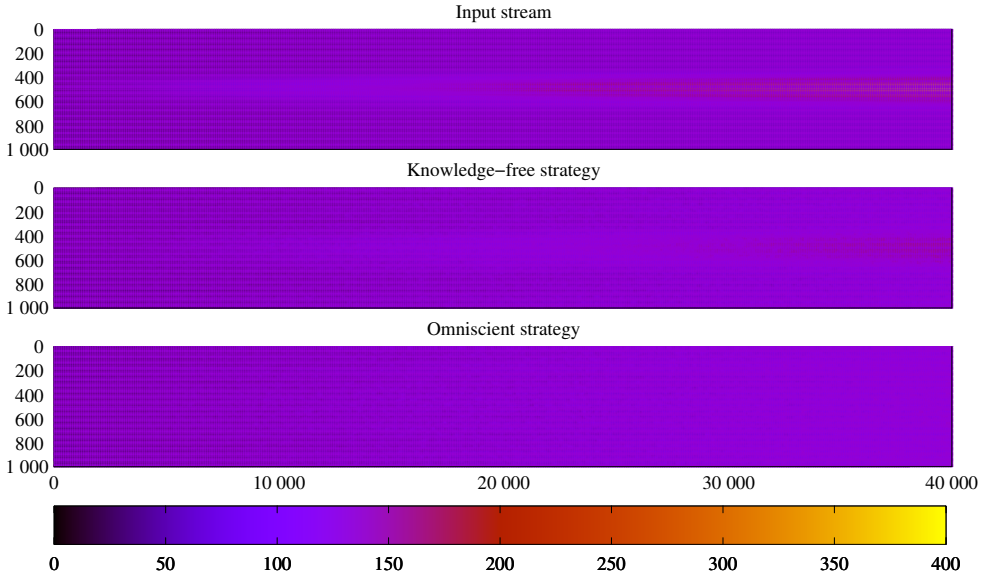


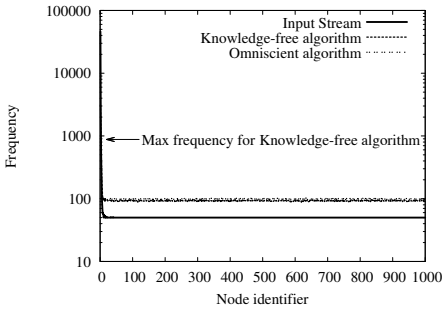
Figure 3.: Frequency distribution as a function of time.
Settings: $n = 1000$, $c = 15$, $s_1 = 10$ and $s_2 = 15$.

distance, the accuracy of our algorithm at node i is computed as

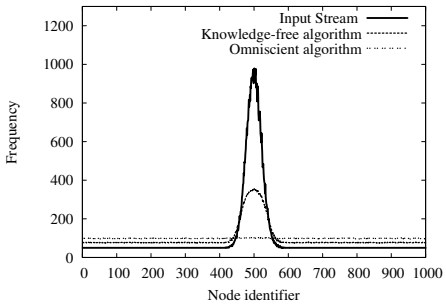
$$G_{ED} = 1 - \frac{\mathcal{D}(S_i, \mathcal{U})}{\mathcal{D}(\sigma_i, \mathcal{U})}.$$

Figure 3 presents a kind of isopleth in which the horizontal axis shows time (or equivalently the number of received node identifiers), the vertical axis represents the node identifiers, and the body of the graph depicts the frequency of each node identifier (*i.e.* the number of occurrences of each node identifier). A lighter color is representative of a very frequent node identifier. The figure at the top of Figure 3 represents the frequency of each node identifier in the input stream of the node sampler. This figure shows that at the inception of the stream, a few number of node identifiers have been received in the input stream which explains the dark color on the left. As time elapses, the number of received identifiers increases (up to $40,000$), and progressively the bias of the input stream appears: a small number of identifiers recur with a high frequency equal to 400, while the frequency of the other node identifiers is significantly lower. This is representative to a truncated Poisson distribution. Now the two other figures represent the output of the node sampler run with respectively the knowledge-free algorithm and with the omniscient one. Clearly the omniscient algorithm succeeds in outputting a uniform stream, illustrated by a color that progressively and uniformly becomes lighter as the number of received identifiers augments. The knowledge-free algorithm is not as good as the omniscient one, nevertheless it succeeds in significantly decreasing the peak of high frequency identifiers with a very small memory (the sampling memory may contain up to 15 node identifiers, and the Count-Min data structure $\hat{F}$ is a 10×15 matrix.) w.r.t. the length m of the input stream.

Figure 4 shows the frequency distribution of node identifiers in respectively the input and output streams as a function of node identifiers. Figure 4a is representative of a particular attack, called *peak attack* in the following, in which the adversary injects



(a) σ is biased by a peak attack generated by Zipfian distribution with $\alpha = 4$.



(b) σ is biased by both targeted and flooding attacks generated by truncated Poisson distribution with $\lambda = n/2$.

Figure 4.: Frequency distribution as a function of node identifiers. Settings: $m = 100,000$, $n = 1,000$, $c = 15$, $s_1 = 10$ and $s_2 = 15$.

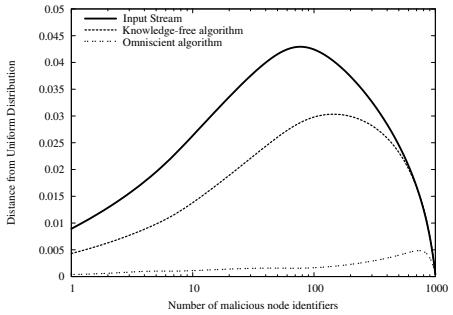
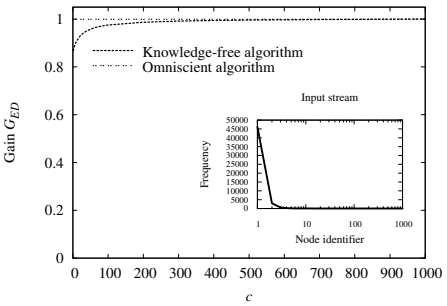
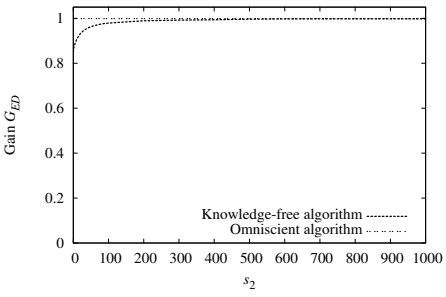


Figure 5.: Euclidean distance as a function of the number of malicious node identifiers. Settings: $m = 100,000$, $n = 1,000$, $c = 15$, $s_1 = 10$ and $s_2 = 15$



(a) G_{ED} as a function of the number of entries c of the sampling memory Γ. The input stream is biased by a peak attack generated by Zipfian distribution with $\alpha = 4$.
Settings: $m = 100,000$, $n = 1,000$, $s_1 = 10$, $s_2 = 15$.

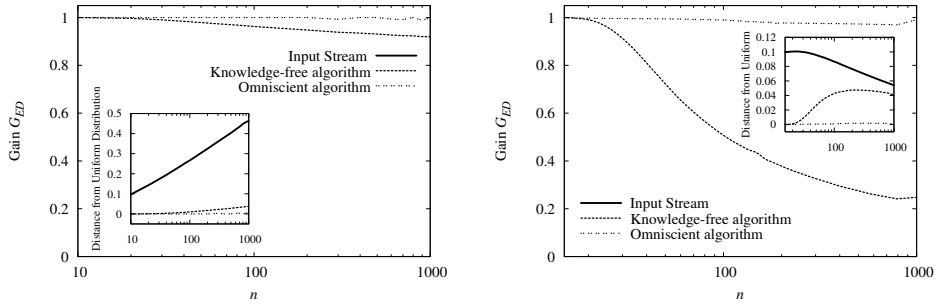


(b) G_{ED} as a function of the number of columns s_2 of the Count-Min Sketch matrix. The input stream is biased by a peak attack generated by Zipfian distribution with $\alpha = 4$.
Settings: $m = 100,000$, $n = 1,000$, $c = 15$, $s_1 = 10$.

Figure 6.: Gain of the knowledge-free algorithm as a function of its parameters.

$50,000$ times a single node identifier while all the other identifiers occur 50 times in the whole stream. Clearly the omniscient algorithm fully tolerates such an attack by successfully outputting a uniform and fresh output stream. The knowledge-free algorithm allows to reduce by a factor 50 the frequency peak with a small amount of memory space with respect to the population size n and the length m of the input stream (the sampling memory contains 15 entries and the Count Sketch matrix contains 150 ones). Figure 4b represents a scenario in which the adversary has successfully subverted the knowledge-free algorithm by launching both a targeted and flooding attacks. Indeed, in this figure around 100 node identifiers are over represented in the input stream σ. Now from Table 1, when $s_2 = 15$, the minimum number of malicious node identifiers that need to be injected by the adversary to make a targeted attack successful with probability 0.9 and 0.9999 is respectively equal to $L_{s_2,s_1} = 68$ and $L_{s_2,s_1} = 168$, while it is equal to $E_{s_2} = 73$ and $E_{s_2} = 173$ to launch a flooding attack. Note that although both attacks are successful, the sampler service divides by 3 the frequencies of malicious node identifiers. Again, the omniscient algorithm is fully robust against both attacks.

Figure 5 complements Figures 4a and 4b by showing the impact of an "uniform" attack on the distance between the output stream and a uniform one as a function of the number of manipulated node ids. More precisely, in these experiments, the adversary progressively takes the control of a uniform input stream by injecting its malicious node ids in such a way that each malicious node identifier appears ten times as many as a correct one. The adversary injects $t = 1, 2, \ldots, 100, \ldots, 1000$ node ids leading to respectively $0.9\%, 1\%, \ldots, 52\%, \ldots, 100\%$ of the total size of the input stream, that is $100,000$ items. The input stream can thus be seen as a combination of two uniform streams, one made of correct node ids, and the other one made of malicious ones, such that the frequency of malicious one is ten times the one of correct node ids. The first observation is the very good behaviour of the omniscient algorithm whatever the proportion of malicious node ids. Now, regarding the knowledge-free algorithm, the worst (but weak) impact is observed when the adversary has succeeded to manipulate 10% of the total number of node identifiers, which amounts to 52% of the input stream. Then the distance slowly decreases until being null, which is easily explained by the fact that the input stream is progressively composed of solely the uniform stream generated by the adversary. While in some way unrealistic, this scenario of attack completes the ones generated with a Zipf with $\alpha = 4$ distribution (*i.e.* peak attack) and a Poisson distribution (targeted and flooding attack) by showing that whatever the effort exerted by the adversary in terms of number of items injected in the input stream, the omniscient algorithm succeeds in outputting a perfectly uniform stream, and the knowledge-free algorithm clearly help to mitigate their impact on the output stream. As an example, Figure 5, and the inset graphs of Figure 7 show that when the adversary floods the input stream with around $50,000$ items, which corresponds *i)* for a uniform attack to send one hundred node ids $5,000$ times each, *ii)* for a peak attack, to flood $48,000$ times a single node id, and *iii)* for a "Poisson" attack to send one hundred nodes ids with a Poisson like frequency distribution, then in all these types of attacks, the distance of the output distribution with a uniform one is very small. Note that a strong incentive for the adversary to generate a specific type of attack is the cost it needs to pay for owning malicious node identifiers. Coming back to the figures, it costs almost zero effort for the adversary to generate a peak attack because it requires a single malicious node id, while in the other two cases the adversary has to interact one hundred times with a central authority to receive certificates assessing the validity ad integrity of the one hundred node identifiers. The good news with our



(a) σ is biased by a peak attack generated by Zipfian distribution with $\alpha = 4$.

(b) σ is biased by both targeted and flooding attacks generated by a truncated Poisson distribution with $\lambda = n/2$.

Figure 7.: G_{ED} as a function of the number of distinct node identifiers n. Settings: $m = 100,000$, $c = 15$, $s_1 = 10$ and $s_2 = 15$.

knowledge-free algorithm is that it is highly efficient in presence of peak attacks.

Figures 6a and 6b illustrate the intuitive fact that increasing the number of entries c of sampling memory, or increasing the available space of the sketch is a very powerful defence against attacks.

Actually, Figure 7 confirms the impressive robustness of the omniscient algorithm, and shows the very good resilience of the knowledge-free algorithm against a peak attack (modelled by a zipfian distribution for Figure 7b). This figure shows the gain G_{ED} of the output stream over the input stream as a function of the number of distinct node identifiers. Note that the inset graph in Figure 7 simply illustrates the ED distance between the input stream and the uniform one (continuous line) and the ED distance between the output stream generated by respectively the omniscient and knowledge-free algorithms and the uniform stream (dotted lines) as a function of the number of node identifiers. This inset graph is obtained with the same parameters setting as the main figure. When the number of distinct ids is close to the size of the sampling memory (*i.e.*, $n \leq 20$), the knowledge-free algorithm builds a uniform distribution since almost all the distinct ids are stored in the sampling memory. Now, for increasing values of n, the distance between the output stream and the uniform one does augment (*i.e.*, G_{ED} decreases) however in a very slight extent. Storing 15% of the node ids in the sampling memory versus 1.5% decreases the gain from 0.95 to 0.9, which shows the good behaviour of the knowledge-free algorithm.

Figure 8 shows the time needed for both output streams (constructed with respectively the omniscient algorithm and the knowledge-free one) to reach their stationary regime. The main observation is that for an input stream made of $1,000$ node identifiers and biased by a peak attack, in between $1,000$ and $10,000$ items must be read by the omniscient and the knowledge-free algorithms to output respectively a uniform and a quasi-uniform stream, which is fully compatible with monitoring requirements.

Now, Figure 9 illustrates the outcome of the sampler service fed by the real traces presented in Section 7.1. As previously said, these real traces share a zipfian behaviour, revealing the presence of a small number of highly frequent node identifiers among a very large number of rare ones. This figure confirms the observations made on synthetic traces, namely the capability for the knowledge-free algorithm to build a quasi uniform stream from a highly biased one in a space efficient way. Note the best results are obtained for the most skewed trace, *i.e.*, the Saskatchewan one, which is explained by

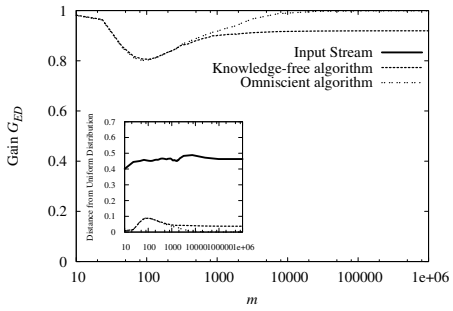


Figure 8.: G_{ED} as a function of the input stream size m. The input stream is biased by a peak attack generated by Zipfian distribution with $\alpha = 4$.
Settings: $n = 1,000$, $c = 15$, $s_1 = 10$ and $s_2 = 15$.

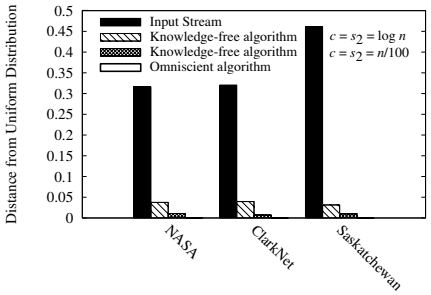


Figure 9.: Euclidean distance between the different streams (input and output ones) and the uniform one. The input stream has been extracted from the real dataset.
Settings: $s_1 = 10$.

the fact that the accuracy of Count-Min sketch is higher in presence of highly frequent items. Regarding the omniscient algorithm, its performance is very good whatever the size c of the sampling memory.

8. Sketches in Series to Build a Uniform Stream

In Section 4.3, we have demonstrated that $q = \min_{j \in \mathcal{N}} p_j$ impacts the time needed for an output stream to converge to a uniform stream (see Theorem 4.3). We have exploited this result by putting sketches in series as illustrated in Figure 10. Experiments have shown that processing the input stream with sketches put in series decreases the convergence time, and this is achieved without requiring any additional space nor additional operations per item. The proposed algorithm, denoted by $\mathcal{A}(r)$ in the following, works as follows. Each node maintains r instances of both data structures $\hat{F}$ and Γ and applies Algorithm 2 on each of these r instances. Upon receipt of some item j from the input stream, $\mathcal{A}(r)$ feeds the first instance of the knowledge-free algorithm with j leading to the output of some item ℓ. This item then feeds the second instance of the knowledge-free algorithm, and the same process is repeated until all the r instances have been traversed. The outputs of $\mathcal{A}(r)$ is the output of the r-th instance of the knowledge-free algorithm. Figure 10 illustrates the construction for $\mathcal{A}(3)$.

We have conducted experiments to validate the performance of our new con-

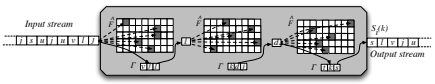
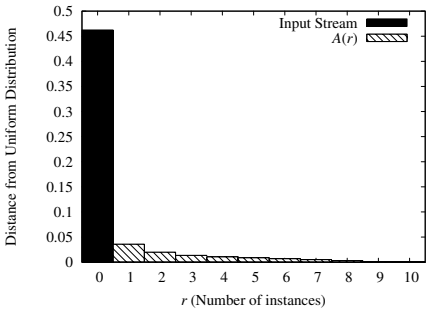
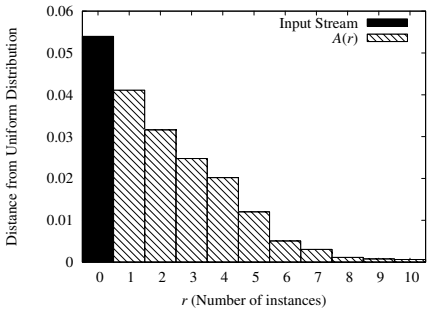


Figure 10.: Algorithm $\mathcal{A}(3)$ run by node $i \in \mathcal{N}$.



(a) σ is biased by a peak attack generated by a Zipf distribution with $\alpha = 4$.



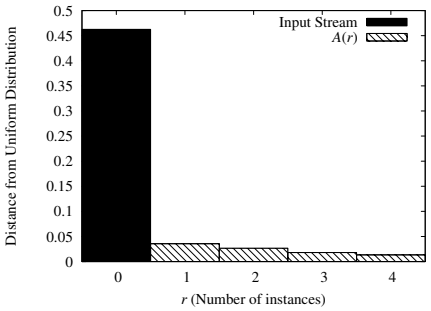
(b) σ is biased by both targeted and flooding attacks generated by a truncated Poisson distribution with $\lambda = n/2$.

Figure 11.: Distance of the output stream generated by $\mathcal{A}(r)$ from the uniform one as a function of the number of instances r of the knowledge-free algorithm. Settings: $m = 100,000$, $n = 1,000$, and $c = 15$, $s_1 = 10$, $s_2 = 15$ for each sketch instance.

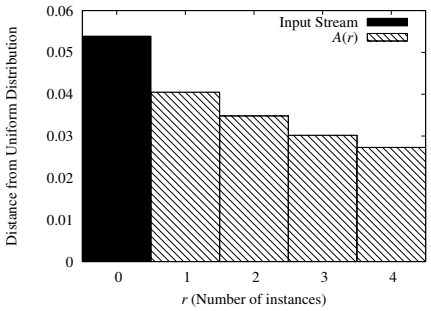
struction. Figure 11 shows that the presence of $r > 1$ sketches, each one using $(s_1 s_2 + c) \log n$ bits of space, improves upon the results obtained with a single sketch of size $(s_1 s_2 + c) \log n$. Figure 12 confirms the interest of such a construction when the total size of the r instances is exactly equal to the size of a single sketch. In order to evaluate the accuracy of our algorithms, we have measured the euclidean distance between the output streams and a uniform one.

In more details, Figure 11 shows the distance between the output stream and a uniform one as a function of the number of instances r of the knowledge-free algorithm, and for two different shapes of input streams. Note that an abscissa equal to zero represents the original input stream. As expected, the output stream built by Algorithm $\mathcal{A}(r)$ gets closer to a uniform stream as a function of r. When the input stream follows a highly skewed distribution (see Figure 11a) the impact of a single instance is predominant compared to the other ones. On the other hand, for less extreme distributions, the improvement is almost proportional to the number of instances. Note however that all these experiments have been conducted by using $(s_1 s_2 + c) \log n$ bits of space for each instance of the knowledge-free algorithm.

Figure 12 shows that $\mathcal{A}(r)$ still outperforms $\mathcal{A}(1)$, even using exactly the same amount of space for both $\mathcal{A}(r)$ and for $\mathcal{A}(1)$. In other words, using a series of small sketches is more efficient than a single but large one to build a uniform stream from an arbitrary one. The last point that needs to be discussed concerned the delay introduced by the presence of r sketches with respect to a single one. Let δ the time needed for a sketch when fed by an item j to output some item ℓ. Then, algorithm $\mathcal{A}(r)$ introduces a delay of δ times r to output its first item, and then the stream is output at the rate of the input stream. On the other hand the presence of r sketches with respect to a single one does not substantially increase the number of operations devoted to each item.



(a) σ is biased by a peak attack generated by a Zipf distribution with $\alpha = 4$.



(b) σ is biased by both targeted and flooding attacks generated by a truncated Poisson distribution with $\lambda = n/2$.

Figure 12.: Distance of the output stream generated by $\mathcal{A}(r)$ from the uniform one as a function of the number of instances r of the knowledge-free algorithm. Settings: $m = 100,000$; $n = 1,000$; and total space used by $\mathcal{A}(r) = (10 \times 15 + 15) \log n$.

9. Conclusion

In this paper, we have studied the node sampling problem in presence of malicious nodes in a very large system by adopting a probabilistic approach. We have proposed and analyzed two online algorithms. The omniscient one is fully resilient to any attacks launched by a strong adversary, while the knowledge-free one is capable of drastically decreasing the impact of adversarial attacks by using small memory space. We are currently revisiting the Count-Min sketch to improve its behaviour when the input distribution shows a long tail, which seems to be popular distributions modeling phenomena where the frequency of input items can vary by many orders of magnitude.

As future work, we plan to extend this work by removing the assumption of a global trusted party supplying verifiable random identifiers. A possible solution would to rely on node stake. The idea would be to assume that nodes own some minimal amount of stake (as for example digital money) that evolves according to node transactions. We might adopt (a simplified version of) what is commonly known as the Bitcoin Unspent Transaction Output (UTXO) model [36]. An UTXO can be roughly seen as a node's account credited by some stake. An UTXO is uniquely characterized by a public key pk_i and its associated amount of stake s_i. Each public key is related to the digital signature schema Σ with the uniqueness property, which allows stakeholders to use the public keys (or a hash thereof) of their UTXOs as a reference to them, as demonstrated in the "Public Keys as Identities principle" of Chaum [37]. Relying on stake has been extensively used to build permissionless blockchains (including Omniledger [38], Ouroboros [39] Algorand [40], Snow White [41], StakeCube [42]). The challenge here would be to find a very efficient way to parse the blockchain or any data structure based on it (such as the UTXO pool) to determine node stake, and thus node legitimacy to appear in the input node stream sampling. By achieving this, we could obtain a fault tolerant random sampling service fully adapted to any large scale system with no trusted third authority.

References

[1] Lv Q, Cao P, Cohen E, et al. Search and Replication in Unstructured Peer-to-Peer Networks. In: Proceedings of the International Conference on Supercomputing (ICS); 2002. p. 84–95.

[2] Bollobás B. Random Graphs – 2nd Edition. Cambridge University Press; 2001.

[3] Demers A, Greene D, Hauser C, et al. Epidemic algorithms for replicated database mangement. In: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC); 1987. p. 1–12.

[4] Jelasity M, Voulgaris S, Guerraoui R, et al. Gossip-based Peer Sampling. ACM Transaction on Computer System. 2007;25(3):1–36.

[5] Stutzbach D, Rejaie R, Duffield NG, et al. On Unbiased Sampling for Unstructured Peer-to-Peer Networks. IEEE/ACM Transactions on Networking. 2009;17(02):377–390.

[6] Jesi GP, Montresor A, van Steen M. Secure Peer Sampling. Computer Networks. 2010; 54(12):2086–2098.

[7] Liu D, Ning P, Du W. Detecting Malicious Beacon Nodes for Secure Location Discovery in Wireless Sensor Networks. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS); 2005. p. 609–619.

[8] Singh A, Ngan TW, Druschel P, et al. Eclipse Attacks on Overlay Networks: Threats and Defenses. In: Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM); 2006. p. 1–12.

[9] Anceaume E, Busnel Y, Gambs S. On the power of the adversary to solve the node sampling problem. Transactions on Large-Scale Data and Knowledge-Centered Systems. 2013;:102–126.

[10] Bortnikov E, Gurevich M, Keidar I, et al. Brahms: Byzantine Resilient Random Membership Sampling. Computer Networks. 2009;53:2340–2359.

[11] Sit E, Morris R. Security Considerations for Peer-to-Peer Distributed Hash Tables. In: Proceedings of the 1rst International Workshop on Peer-to-Peer Systems (IPTPS). Springer-Verlag; 2002. p. 261–269.

[12] Douceur J, Donath J. The Sybil Attack. In: Proceedings of the 1rst International Workshop on Peer-to-Peer Systems (IPTPS); 2002. p. 251–260.

[13] Awerbuch B, Scheideler C. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In: Proceedings of the 31rst International Colloquium on Automata, Languages and Programming (ICALP); 2004. p. 183–195.

[14] Awerbuch B, Scheideler C. Towards a Scalable and Robust Overlay Network. In: Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS); 2007.

[15] Anceaume E, Ludinard R, Sericola B, et al. Modeling and evaluating targeted attacks in large scale dynamic systems. In: Proceedings of the 41rst IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2011.

[16] Anceaume E, Busnel Y, Rivetti N, et al. Identifying global icebergs in distributed streams. In: Proceedings of the 35th IEEE Symposium on Reliable Distributed Systems (SRDS'15); Sep.; Montreal, Canada; 2015.

[17] Rivetti N, Anceaume E, Busnel Y, et al. Online Scheduling for Shuffle Grouping in Distributed Stream Processing Systems. In: Proceedings of the ACM/IFIP/USENIX Middleware 2016; 2016.

[18] Muthukrishnan SM. Data streams: Algorithms and applications. Now Publishers Inc.; 2005.

[19] Bar-Yossef Z, Jayram TS, Kumar R, et al. Counting distinct elements in a data stream. In: Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM). Springer-Verlag; 2002. p. 1–10.

[20] Flajolet P, Martin GN. Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences. 1985;31(2):182–209.

[21] Kane DM, Nelson J, Woodruff DP. An optimal algorithm for the distinct element problem. In: Proceedings of the Symposium on Principles of Databases (PODS); 2010. p. 41–52.

[22] Alon N, Matias Y, Szegedy M. The space complexity of approximating the frequency moments. In: Proceedings of the 28th annual ACM symposium on Theory of computing (STOC); 1996. p. 20–29.
[23] Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. Theoretical Computer Science. 2004;312(1):3–15.
[24] Chakrabarti A, Cormode G, McGregor A. A near-optimal algorithm for computing the entropy of a stream. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms; 2007. p. 328–335.
[25] Lall A, Sekar V, Ogihara M, et al. Data streaming algorithms for estimating entropy of network traffic. In: Proceedings of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS). ACM; 2006. p. 145–156.
[26] Anceaume E, Busnel Y. Lightweight metric computation for distributed massive data streams. Transactions on Large-Scale Data and Knowledge-Centered Systems. 2017;33:1–33.
[27] Anceaume E, Busnel Y. Deviation estimation between distributed data streams. In: Proceddings of the 10th European Dependable Computing Conference (EDCC); 2014.
[28] Lynch N. Distributed algorithms. Morgan Kaufmann Publishers; 1996.
[29] Godfrey PB, Shenker S, Stoica I. Minimizing churn in distributed systems. In: Proceedings of the ACM SIGCOMM; 2006. p. 147–158.
[30] Vitter J. Random sampling with a reservoir. ACM Transactions on Mathematical Software. 1985;37(57).
[31] Anceaume E, Busnel Y, Sericola B. Uniform node sampling service robust against collusions of malicious nodes. In: Proceedings of the 43rst IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2013.
[32] Keneny JG, Snell JL. Finite markov chains. Springer-Verlag; 1976.
[33] Rubino G, Sericola B. On weak lumpability in Markov chains. Journal of Applied Probability. 1989;26:446–457.
[34] Cormode G, Muthukrishnan SM. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms. 2005;55(1):58–75.
[35] Archive TIT. http://ita.ee.lbl.gov/html/traces.html [Lawrence berkeley national laboratory]; Online on feb. 2016.
[36] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system [https://bitcoin.org/bitcoin.pdf]; 2008.
[37] Chaum D. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM. 1988;24(2):84–90.
[38] Kokoris-Kogias E, Jovanovic P, Gasser L, et al. Omniledger: A secure, scale-out, decentralized ledger via sharding. In: IEEE Symposium on Security and Privacy (SSP); 2018.
[39] Badertscher C, Gaži P, Kiayias A, et al. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: ACM SIGSAC Conference on Computer and Communications Security (CCS); 2018.
[40] Gilad Y, Hemo R, Micali S, et al. Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles; ACM; 2017. p. 51–68.
[41] Daian P, Pass R, Shi E. Snow White: Provably Secure Proofs of Stake [Cryptology eprint archive, report 2016/919]; 2016. https://eprint.iacr.org/2016/919.
[42] Durand A, Hbert G, Toumi K, et al. The stakecube blockchain : Instantiation, evaluation and applications. In: Proceedings of IEEE International Conference on Blockchain Computing and Applications (BCCA); 2020.